Handwritten Digit Recognition using Machine Learning

1) Objective:

The objective of this project is to develop a machine learning model that can accurately recognize and predict handwritten digits (0-9) from the MNIST dataset. This model will be trained to classify images of handwritten digits and evaluated based on its prediction accuracy.

2) Data Source:

The dataset used for this project is the MNIST dataset. It contains 70,000 grayscale images of handwritten digits, split into 60,000 training images and 10,000 test images. Each image is of size 28x28 pixels, representing digits 0 to 9.The dataset can be found on:Kaggle MNIST Dataset: https://www.kaggle.com/c/digit-recognizere

TensorFlow/Keras MNIST: This can be imported directly from TensorFlow/Keras libraries.

3) Import Libraries:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, confusion_matrix

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten
```

4) Load the Data:

```
# Load the MNIST dataset

(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

5) Describe Data:

The MNIST dataset consists of:

60,000 training images and 10,000 test images.

Each image is a 28x28 pixel grayscale image.

Each image corresponds to a label, which is an integer between 0 and 9, representing the digit in the image.

```
# Display the shape of the dataset
print("Training Data Shape:", X_train.shape)
print("Testing Data Shape:", X_test.shape)
print("Sample label:", y_train[0])
# Display some images from the dataset
plt.figure(figsize=(8,8))
for i in range(9):
    plt.subplot(3, 3, i+1)
    plt.imshow(X_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.show()
```

6) Data Visualization:

We can visualize some of the digits in the dataset to get a better understanding of the data.

```
# Visualizing few random images
plt.figure(figsize=(10,10))
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(X_train[i], cmap='gray')
    plt.title(f"Digit: {y_train[i]}")
    plt.axis('off')
plt.show()
```

7) Data Preprocessing:

Rescale the pixel values to a range between 0 and 1 by dividing by 255, as neural networks generally perform better with normalized data.

Convert labels to categorical format for classification tasks.

```
# Normalize the pixel values
X_train = X_train / 255.0
```

X_test = X_test / 255.0

Convert labels to categorical (One-hot encoding)

y_train_cat = to_categorical(y_train, 10)

y_test_cat = to_categorical(y_test, 10)

8) Define Target Variable (y) and Feature Variables (X):

X: The images, which are represented by 28x28 pixel grids (feature variables).

y: The labels (0-9) that represent the digit in the image (target variable).

In this case, after preprocessing, X_train and X_test are the feature variables, while y_train_cat and y_test_cat are the target variables in one-hot encoded form.

9) Train-Test Split:

Since the MNIST dataset already provides training and test sets, no additional split is needed. However, we can still create a validation set from the training data to fine-tune the model.

X_train, X_val, y_train_cat, y_val_cat = train_test_split(X_train, y_train_cat, test_size=0.1, random_state=42)

10) Modeling:

Here, we use a simple neural network model for digit recognition.

```
# Build the model

model = Sequential([

    Flatten(input_shape=(28, 28)),  # Flatten the 28x28 images into 1D vectors

    Dense(128, activation='relu'),  # First hidden layer with 128 neurons

    Dense(64, activation='relu'),   # Second hidden layer with 64 neurons

    Dense(10, activation='softmax') # Output layer for 10 digit classes

])

# Compile the model

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X_train, y_train_cat, validation_data=(X_val, y_val_cat), epochs=10, batch_size=32)
```

11) Model Evaluation:

Evaluate the performance of the model on the test set.

```
# Evaluate the model on the test set

test_loss, test_acc = model.evaluate(X_test, y_test_cat)
```

```python
print(f"Test Accuracy: {test_acc*100:.2f}%")

# Confusion Matrix

y_pred = model.predict(X_test)

y_pred_classes = np.argmax(y_pred, axis=1)

y_true = np.argmax(y_test_cat, axis=1)

conf_matrix = confusion_matrix(y_true, y_pred_classes)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.show()
```

12) Prediction:

You can now predict on new images or specific samples from the test set.

```python
# Predict on the first test image

plt.imshow(X_test[0], cmap='gray')

plt.show()

pred = np.argmax(model.predict(X_test[0].reshape(1, 28, 28)), axis=1)

print(f"Predicted Digit: {pred[0]}")
```

13) Explanation:

The project involved building a simple neural network for classifying handwritten digits using the MNIST dataset. After loading and preprocessing the data, the model was trained using fully connected layers, evaluated on a test set, and achieved a reasonable accuracy. The confusion matrix was used to visualize the performance of the model across different classes.